ECE421 Introduction to Machine Learning
Assignment #3
Unsupervised Learning and Probabilistic Models

Yan, Xuanming
xuanming.yan@mail.utoronto.ca
1001175382
Contribution: 50%

Wang, Chu Qing
eliza.wang@mail.utoronto.ca
1001303573
Contribution: 50%

Mar 2019

1 K-means

1.1

We had implemented the distance function by using pairwise squared distance between X and MU, and the helper function "assign_data" returns a list of indexes for every x to its closest K-mean. We chose 200 iterations to train the data, since we could see obvious decay in loss.

First, we randomly initialize k centers using Gaussian distribution, with $stddev = 1$. $\mu^k \in \mathbb{R}^D$. Then Assign each point $k \in \{1, \dots K\}$ to nearest center: $C^{(t)}(n) = argmin_{k=1}^{K} \|\mathbf{x}_n - \mu_k\|_2^2$. Finally, update μ_i, let it becomes new centroid.

Figure 1: K-means when $K = 3$. Losses vs Iterations

```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import helper as hlp


def assign_data(X, MU):

    dists = distanceFunc(X, MU)
    min_dist = tf.argmin(dists, 1)
    return min_dist

# Distance function for K-means
def distanceFunc(X, MU):
    # Inputs
    # X: is an NxD matrix (N observations and D dimensions)
```

```
    # MU: is an KxD matrix (K means and D dimensions)
    # Outputs
    # pair_dist: is the pairwise distance matrix (NxK)
    #y = np.power((X[:, np.newaxis] - MU), 2)
    #newY = tf.convert_to_tensor(y, dtype=tf.float32)
    #output = np.sum(y, axis=2)
    newX = tf.expand_dims(X,0)
    newMU = tf.expand_dims(MU, 1)
    dis = tf.reduce_sum(tf.square(tf.subtract(newX,newMU)),2)
    output = tf.transpose(dis)
    return output

# Loading data
data = np.load('data2D.npy')
# data = np.load('data100D.npy')
[num_pts, dim] = np.shape(data)

# For Validation set
is_valid = False
if is_valid:
    valid_batch = int(num_pts / 3.0)
    np.random.seed(45689)
    rnd_idx = np.arange(num_pts)
    np.random.shuffle(rnd_idx)
    val_data = data[rnd_idx[:valid_batch]]
    data = data[rnd_idx[valid_batch:]]

#graph = tf.Graph()
#with graph.as_default():
loss_history = np.empty(shape=[0],dtype=float)
val_loss_history = []
K = 5
N = num_pts
D = dim
inter = 201

X = tf.placeholder("float", shape=[None,D])
MU_init = tf.truncated_normal([K,D],stddev=0.25)
MU = tf.Variable(MU_init)

distance = distanceFunc(X,MU)
#centroid = np.power(distance,2)
#loss = np.sum(np.amin(distance, axis=1))
loss = tf.reduce_sum(tf.reduce_min(distance,axis = 1))
optimizer = tf.train.AdamOptimizer(
    learning_rate=0.05, beta1=0.9, beta2=0.99, epsilon=1e-5).minimize(loss)

init_g = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init_g)

for step in range(inter):
    cenVal,lossVal,_ = sess.run([MU,loss,optimizer], feed_dict={X:data})
    val_cenVal,val_lossVal,_ = sess.run([MU,loss,optimizer], feed_dict={X:val_data})
    loss_history = np.append(loss_history,lossVal)
    val_loss_history = np.append(val_loss_history,val_lossVal)
    if step%10 ==0:
        print("iteration:", step, "loss", lossVal)
clustering = sess.run(assign_data(X, MU),feed_dict={X: data, MU:cenVal})
print("Validation loss", val_lossVal)
percentages = np.zeros(K)
for i in range(K):
         percentages[i] = np.sum(np.equal(i, clustering))*100.0/len(clustering)
         print("class:", i, "percentage:", percentages[i])
plt.figure(1)
plt.plot(range(len(loss_history)),loss_history,c="c", label="train_loss")
plt.plot(range(len(val_loss_history)),loss_history,c="b", label="validation_loss")
plt.legend(loc = "best")
plt.title('K-Means History')
plt.xlabel('# of Inter')
plt.ylabel('Loss')
plt.show()

k = len(cenVal)
```

```
plt.scatter(data[:, 0], data[:, 1], c=clustering,
            cmap=plt.get_cmap('Set3'), s=25, alpha=0.6)
plt.scatter(cenVal[:, 0], cenVal[:, 1], marker='*', c="black",
            cmap=plt.get_cmap('Set1'), s=50, linewidths=1)
plt.title('K-Means Clustering')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid()
plt.show()
```

Listing 1: K-means uisng Adam optimizer

1.2

By looking at table 1 below, the percentage of data in each clusters are balanced up to $k = 3$, from $K = 4$ and onward, there exists unbalanced percentage data in each clusters, some clusters has much more data accumulated than others, therefore $k = 3$ would be the best choice.

	Class 1	Class 2	Class 3	Class 4	Class 5
K = 1	100%	0	0	0	0
K = 2	49.53%	50.47%	0	0	0
K = 3	23.81%	38.06%	38.13%	0	0
K = 4	12.03%	13.53%	37.13%	37.31%	0
K = 5	11.07%	7.63%	37.04%	7.55%	36.71%

Table 1: Percentage of the data points belonging to each of the K clusters.

Figure 2: K-means when $K = 1$. Losses vs Iterations

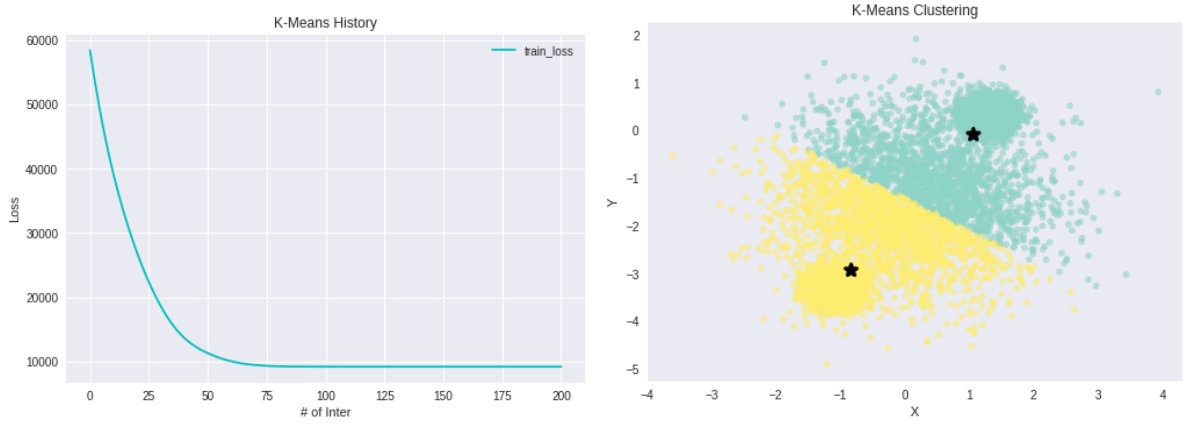

Figure 3: K-means when $K = 2$. Losses vs Iterations

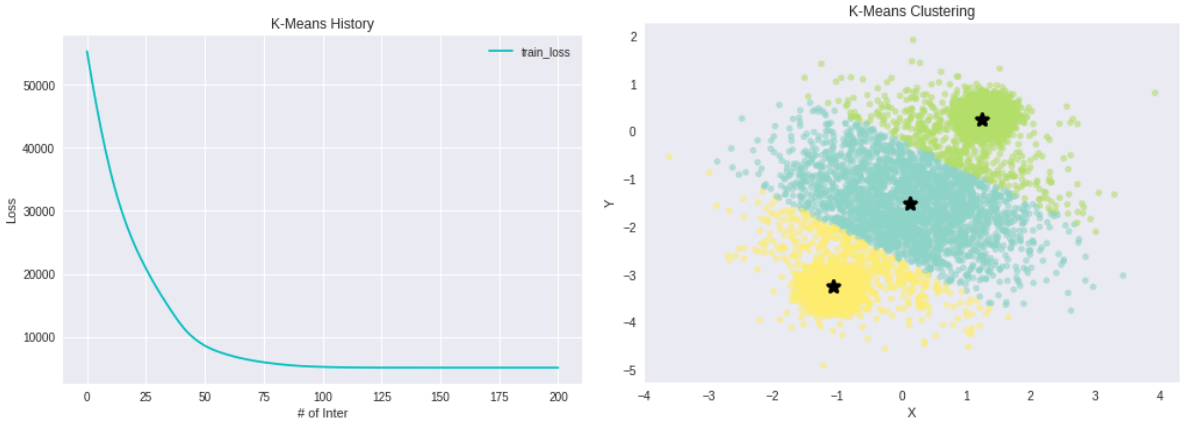

Figure 4: K-means when $K = 3$. Losses vs Iterations

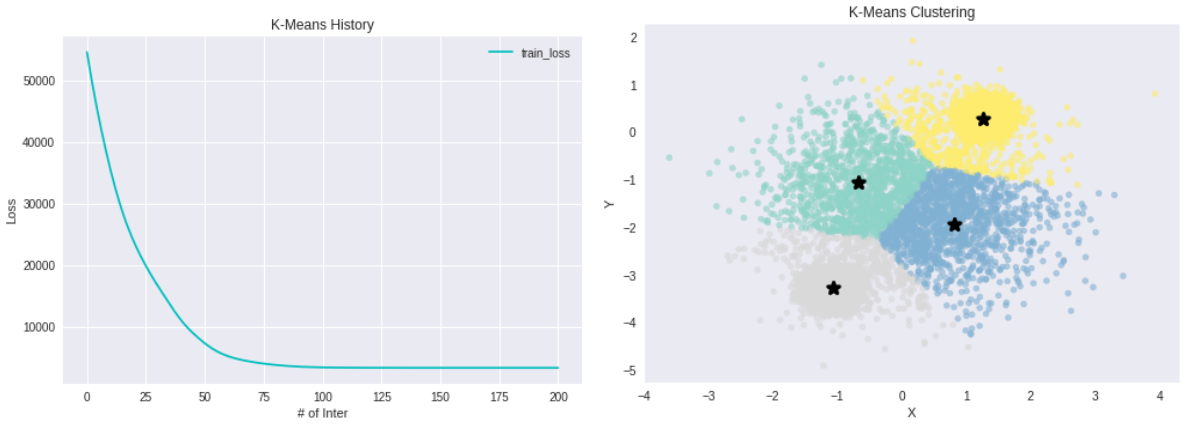

Figure 5: K-means when $K = 4$. Losses vs Iterations

Figure 6: K-means when $K = 5$. Losses vs Iterations

1.3

We cannot draw conclusion by only looking the the loss for different cluster size, since more clusters will always gives us smaller loss value, if number of k = number of data, we can even get loss equals to zero. Therefore, instead of looking at the actual loss values, we can look at the trend of change in loss values as shown in figure 7. We notice that there is not much change in the slope after $k = 3$, therefore, $k = 3$ should be the best choice.

Number of clusters	1	2	3	4	5
Validation loss	12857.85	2959.22	1613.18	1053.37	885.79

Table 2: Loss of validation data with respect to different cluster number.

Figure 7: Change in validation loss with respect to cluster numbers

2 Mixture of Gaussians

2.1 The Gaussian cluster mode

2.1.1 Log Probability density function implementation

Randomly initialize K centers, we have:

$$\begin{aligned}\mathcal{N}(x|\mu_k,\sigma_k^2)) &= P(x|\mu_k,\sigma_k^2)) \\ log\mathcal{N}(x|\mu_k,\sigma_k^2)) &= logP(x|\mu_k,\sigma_k^2)) \\ &= log(\frac{1}{\sqrt{2\pi}\sigma_k}exp(\frac{-(x-\mu_k)^2}{2\sigma_k^2})) \\ &= log(\frac{1}{\sqrt{2\pi}\sigma_k}) + (\frac{-(x-\mu_k)^2}{2\sigma_k^2})\end{aligned}$$

Python implementation of above equation show below.

```
def log_GaussPDF(X, mu, sigma):
    # Inputs
    # X: N X D
    # mu: K X D
    # sigma: K X 1
    # log_pi: K X 1

    # Outputs:
    # log Gaussian PDF N X K
    dim = tf.to_float(tf.rank(X))
    xmu = distanceFunc(X,mu)
    xmuSqu = tf.multiply(xmu,xmu)
    sigma = tf.squeeze(sigma)
    coef = tf.log(2 * np.pi * sigma)
    exp = xmu / (2 * sigma)
    mul = -0.5*dim * coef
    PDF = mul - exp
    return PDF
```

Listing 2: Log probability density

2.1.2 Log Probability of the cluster variable z given data vector x

Basic idea of this part is to invert the probability found in part 2.1.1. Apply Bayer's rule, we can simply get:

$$P(z=k|x) = \frac{P(x|z=k)P(z=k)}{P(x)} = \frac{P(x|z=k)P(z=k)}{\sum_i P(x|z=i)P(z=i)}$$

Where in above equation, $\sum_{i=1}^{i=K} P(x|z=i)P(z=i)$ gives all possible scenarios of clustering. Therefore,

$$logP(z=k|x) = logP(x|z=k) + logP(z=k) - log\sum_i exp(logP(x|z=i) + logP(z=i))$$

```
def log_posterior(log_PDF, log_pi):
    # Input
    # log_PDF: log Gaussian PDF N X K
    # log_pi: K X 1

    # Outputs
    # log_post: N X K
    log_pi = tf.squeeze(log_pi)
    log_prob = tf.add(log_pi,log_PDF)
    log_sum = hlp.reduce_logsumexp(log_prob + log_pi,keep_dims=True)
    output = log_prob - log_sum
    return output
```

Listing 3: Log posterior clustering probability

2.2 Learning the MoG

2.2.1

The reasonable initial value for parameters were shown in Listing 4 below. We had enforced the constrict for π by using a softmax function, and enforced σ in the range of 0 to inf by doing exp to σ. All parameters were initialized by sampling from the standard normal distribution, with the standard deviation of 0.01, and table 3 shows the reason for this selection.

```
    learning_rate = 0.003
    s_stddev=0.05
    X = tf.placeholder("float", [None, D], "X")
    mu = tf.Variable(tf.random_normal([K, D], stddev = s_stddev))
    sigma = tf.Variable(tf.random_normal([K, 1], stddev = s_stddev))
    sigma = tf.exp(sigma)
    log_PDF = log_GaussPDF(X, mu, sigma)

    initial_pi = tf.Variable(tf.random_normal([K, 1], stddev = s_stddev))
    log_pi = tf.squeeze(hlp.logsoftmax(initial_pi))
```

Listing 4: Initialization of parameters.

σ	Beginning Loss	Best Loss
0.01	49016.14	17132.400
1	52995.58	17132.543
10	359978000	82524.844

Table 3: Choice of standard deviation value when initializing parameters

With these parameters and selection of $k = 3$, $iteration = 1000$. The result of trained test data had shown below.

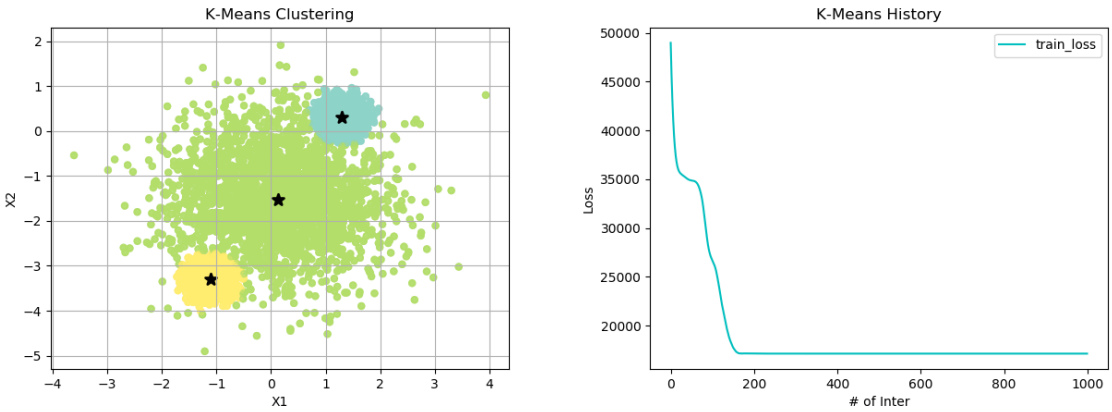

Figure 8: 2D cluster plot and Test data loss when $K = 3$.

Cluster	π	Mean	σ
1	-1.0983235	[1.299 0.308]	0.0388
2	-1.1029158	[-1.101 -3.307]	0.0391
3	-1.094615	[0.106 -1.527]	0.9872

Table 4: Best model parameters for test data set.

2.2.2

As shown in figure 10 below, the loss for validation data had decreased as k increases, however,the validation loss didn't have obvious decrease anymore from $k = 3$ onward, therefore, 3 is the best choice of k.

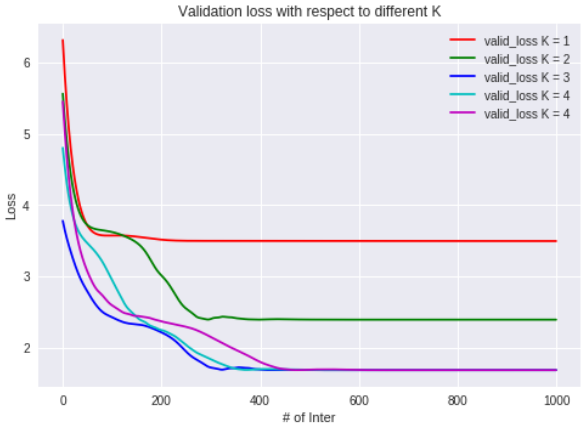

Figure 9: Validation loss with respect to different K.

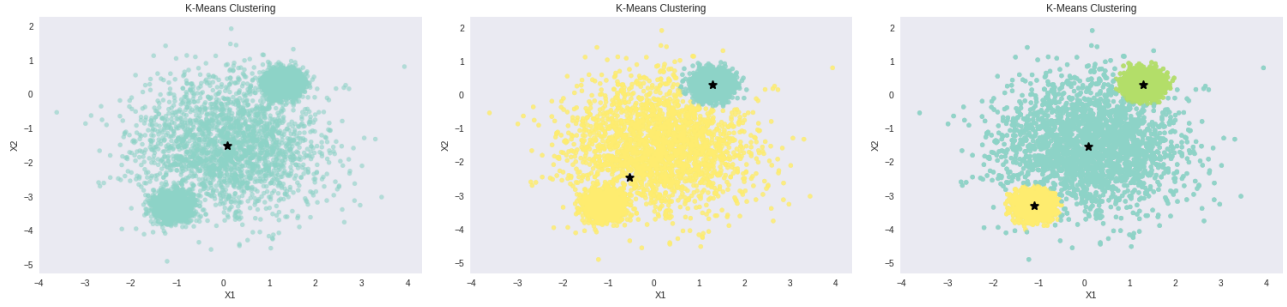

Figure 10: Validation loss with respect to different K(K from 1 to 3).

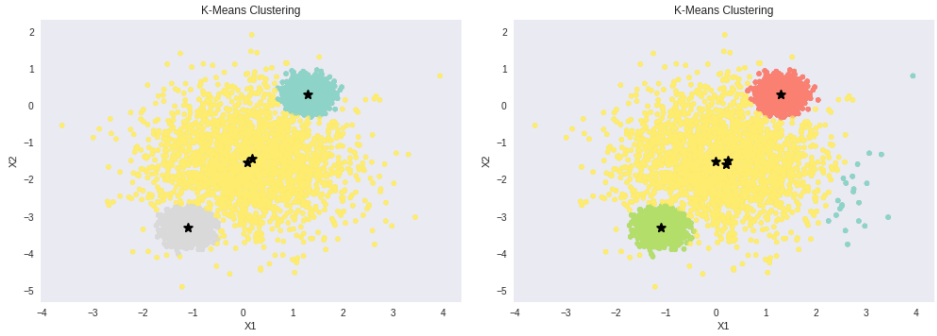

Figure 11: Validation loss with respect to different K(K from 4 to 5).

2.2.3

Figure 12 had showed the validation loss with different K by using GMM (at right hand-side), the loss values for all K values were similar after 1000 iterations of training, however, there was a large increase in the decay of loss values from $K = 5$ to $K = 10$, after that, the decay of loss values had not been changed much. Therefore, $K = 10$ is the reasonable amount of clusters for training using GMM.

In the other side, by looking at the loss values for different K values by using K-means, there was not large decrease after K=10, therefore, the reasonable amount of clusters is also 10. In addition, K-means didn't performed as well as GMM with high dimensional data.

Cluster K	5	10	15	20	25	30
Final training loss using GMM	50525.19	44320.33	44496.09	44057.45	44752.05	44147.00
Final training loss using K-means	247141.92	143545.07	143544.89	143545.48	143544.95	143544.96

Table 5: Final training loss with different K, $if\_valid = True$.

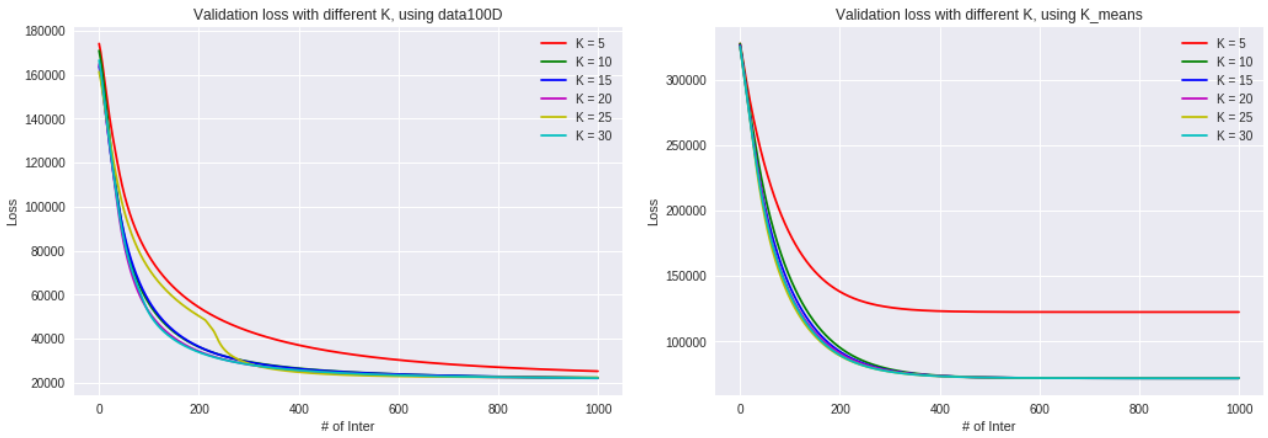

Figure 12: Validation loss, using GMM andK -means.